Défis dans le développement collaboratif et ouvert de l'assistant de preuve Coq et de son écosystème

Théo Zimmermann[1,2] theo@irif.fr

[1] Université de Paris, IRIF, CNRS, F-75013 Paris, France
[2] Inria, équipe-projet πr^2

1 Introduction

Coq [1] est un logiciel libre développé par des chercheurs Inria depuis plus de 35 ans, qui permet d'énoncer formellement des théorèmes mathématiques et d'en vérifier les preuves. Ce logiciel et le langage associé s'appuient sur des décennies de recherche sur la logique d'une part, et les langages de programmation fonctionnels d'autre part. Cependant, à mesure que ce logiciel grandit, en nombre d'utilisateurs, en taille et complexité du code, et que son développement s'ouvre davantage aux contributeurs extérieurs, d'autres types de questions de recherche se posent.

Ces nouvelles questions ont trait notamment à la maintenance et à l'évolution du logiciel, à la facilitation de son installation et de son utilisation, à l'implication des utilisateurs dans le développement open source du logiciel, sa documentation, et son écosystème de paquets.

La thèse que je défends dans mon travail de doctorat est que le succès d'un logiciel mathématique complexe et d'un langage fonctionnel fortement typé tel que Coq ne pourra être effectif que grâce aux avancées que permettront le mariage de la théorie et de la pratique, de la recherche en logique et langages de programmation, avec celle en génie logiciel. Historiquement, cette dernière a davantage été associée aux langages orientés objet, qui en ont beaucoup bénéficié. Dans le même temps, les langages fonctionnels, plus à la pointe au niveau des concepts, et permettant de programmer de manière plus sûre et abstraite, pâtissaient grandement d'un manque d'outillage avancé.

2 Démarche

Mon objet d'étude est un développement logiciel spécifique, et ma démarche consiste à identifier les problèmes concrets dont souffrent les développeurs ou la communauté d'utilisateurs en étant moi-même impliqué au cœur du développement. Une fois les problèmes concrets identifiés, il est nécessaire de prendre du recul afin d'en déterminer la substance. Cette étape d'abstraction permet de raisonner sur une question qui n'est plus spécifique à ce développement logiciel ou cette communauté d'utilisateurs, même si des particularismes peuvent continuer à jouer un rôle dans la mise en place de solutions génériques.

Parfois, les solutions à ces problèmes ont déjà été clairement identifiées et n'ont plus qu'à être appliquées ; parfois la littérature scientifique n'a pas encore abordé le problème en question mais d'autres projets open source ont été confrontés à un problème similaire et ont proposé une solution. Mais l'évolution rapide et continue des processus et outils de développement dans le monde du logiciel libre fait que les problèmes sont parfois nouveaux, ou trop spécifiques, et sont en attente de solutions novatrices. Dans ce qui suit, j'illustre cette démarche sur des exemples concrets.

Mesurer l'impact d'un changement de bug tracker. Le logiciel Coq est aussi ancien que le concept même de logiciel libre. Le dépôt contenant son code source versionné date pour sa part de 1999, année de création du terme « open source ». En 2007, c'est-à-dire juste avant la création de GitHub, le projet avait commencé à utiliser le *bug tracker* Bugzilla, l'un des bug trackers libres les plus connus, utilisé par de nombreux logiciels libres importants. Dix ans plus tard, en 2017, GitHub était devenu la plateforme centrale de développement de Coq. Le système de *pull request* était utilisé, non seulement pour intégrer les contributions extérieures, mais aussi pour tester et valider les changements proposés par les développeurs principaux. Dans ce contexte, l'utilisation de Bugzilla

devenait de plus en plus problématique : nécessité de changer de plateforme pour naviguer entre les tickets et le code, lourdeur, lenteur, et manque de maniabilité comparative, qui rendaient son utilisation plus difficile, à la fois pour les développeurs principaux, et les contributeurs extérieurs.

J'ai tout d'abord émis l'idée d'opérer un basculement vers le bug tracker intégré à GitHub, en proposant un comparatif des fonctionnalités. Puis, j'ai démontré la faisabilité d'une migration des tickets pré-existants en réutilisant et adaptant un outil [2] qui n'était pas conçu pour supporter une migration de l'ampleur de celle de Coq. Cette évaluation de la faisabilité et des bénéfices a permis à l'équipe de développement de valider le changement de bug tracker, que j'ai alors effectué.

Enfin, à l'aide de données de l'activité sur le bug tracker dans une période de plus de 18 mois avant et autant après le changement, et de la méthode de *Regression on Discontinuity*, nouvelle en génie logiciel empirique, j'ai montré [3] l'effet causal de ce changement de bug tracker sur l'activité : les développeurs y sont plus actifs et les non-développeurs sont plus nombreux à participer aux discussions sur les tickets. Ce résultat est basé sur un seul cas d'étude et devrait être répliqué sur d'autres projets, mais il est aussi la première comparaison de bug trackers du genre.

Gérer la documentation des changements dans les nouvelles versions. En 2017, j'ai mis en place un nouveau processus d'intégration des pull requests basé sur le modèle (répandu) du *backporting*, avec un *release manager* qui prend les décisions de backporter ou non chaque changement, alors qu'auparavant les développeurs devaient choisir eux-mêmes pour quelle branche (stable ou *master*) ils préparaient leurs correctifs (les branches stables étant fusionnées plus tard dans *master*). Or, cette simplification du processus a pour conséquence qu'au moment de la préparation et de l'intégration d'un correctif, les développeurs ne savent pas avec certitude dans quelle version celui-ci sera présent en premier, ce qui a généré des incohérences dans le fichier de *changelog*.

La solution que j'ai proposée est inspirée par une idée des développeurs de GitLab [4] pour résoudre les conflits récurrents dans leur changelog. Chaque pull request décrit les changements qu'elle introduit dans un fichier séparé, et c'est au moment de sortir une nouvelle version que la documentation des changements dans cette version est compilée, sur la base du sous-ensemble de ces fichiers se trouvant dans la branche stable correspondant.

Alors que la plupart des recherches sur le backporting ont porté sur l'exemple du noyau Linux, mon analyse détaillée des différentes options méthodologiques adaptées à un projet de taille moyenne, et de leurs limites, devrait pouvoir bénéficier à de nombreux projets.

S'assurer de la maintenance des paquets sur le long terme. La force d'un *framework* ou d'un langage de programmation réside souvent en grande partie dans son écosystème de paquets. Les paquets développés pour Coq ont la double particularité d'avoir fortement besoin d'être maintenus pour rester compatibles avec les évolutions de Coq, et d'être souvent l'œuvre d'étudiants, notamment en thèse, qui ne restent pas toujours actifs dans la communauté sur le long terme.

L'initiative *coq-community* [5], que j'ai lancée en m'inspirant d'*elm-community* [6], est une organisation informelle d'utilisateurs dédiée à la maintenance sur le long terme de paquets Coq suscitant l'intérêt. Moins d'un an après son lancement en juillet 2018, elle connait des débuts prometteurs puisqu'elle accueille déjà 16 paquets et projets, maintenus par 11 mainteneurs principaux. Cet effort est aussi une occasion de proposer et de tester des outils et des processus nouveaux dans la maintenance de paquets Coq, et de contribuer à des investigations plus générales sur la comparaison de divers écosystèmes de paquets en fonction de certaines caractéristiques structurantes.

Références

1. The Coq development team: Coq 8.9.0 (2019). https://doi.org/10.5281/zenodo.1003420.
2. Berestovskyy, A.: bugzilla2github, `https://github.com/berestovskyy/bugzilla2github/`.
3. Zimmermann, T., Casanueva Artís, A.: Impact of switching bug trackers: a case study on a medium-sized open source project. In *International Conference on Software Maintenance and Evolution*, 2019.
4. Speicher, R.: How we solved GitLab's CHANGELOG conflict crisis, `https://about.gitlab.com/2018/07/03/solving-gitlabs-changelog-conflict-crisis/`. Last accessed 17 May 2019.
5. Coq-community manifesto, `https://github.com/coq-community/manifesto`.
6. Elm-community homepage, `http://elm-community.org`. Last accessed 17 May 2019.